MODULE *ISSU*

EXTENDS *Naturals*, *FiniteSets*, *Sequences*, *TLC*

The set of all nodes
CONSTANT *Nodes*

A sequence of all versions
CONSTANT *Versions*

Node states
CONSTANT *Started*, *Stopped*

Upgrade states
CONSTANTS *Inactive*, *Initialized*, *Upgraded*, *Committed*, *RolledBack*, *Reset*

The current state of the upgrade
VARIABLE *upgradeState*

The current upgrade version
VARIABLE *upgradeVersion*

The set of node states
VARIABLE *nodes*

The number of versions must be greater than 1
ASSUME $Len(Versions) > 1$

The number of nodes must be greater than 1
ASSUME $Cardinality(Nodes) > 1$

The current version domain
$CurrentVersion(v) \triangleq$ CHOOSE $i \in$ DOMAIN $Versions : Versions[i] = v$

The next version
$NextVersion(v) \triangleq Versions[CurrentVersion(v) + 1]$

The previous version
$PreviousVersion(v) \triangleq Versions[CurrentVersion(v) - 1]$

Predicate indicating whether the version can be incremented
$CanUpgrade(v) \triangleq$
$\quad \land CurrentVersion(v) < Len(Versions)$

Helper predicate indicating whether an upgrade can be rolled back during a node state change
$CanRollback \triangleq$
$\quad \land upgradeState = Upgraded$
$\quad \land Cardinality(\{n \in Nodes : nodes[n].version = PreviousVersion(upgradeVersion)\}) > 0$

Checks that there's never more than two versions running in the cluster at a given time and that only one version may be running if the upgrade is *Inactive*.

$$
\begin{array}{l}
TypeInvariant \triangleq \\
\quad \land \lor \land upgradeState = Inactive \\
\qquad \land Cardinality(\{nodes[n].version : n \in Nodes\}) = 1 \\
\quad \lor \land upgradeState \neq Inactive \\
\qquad \land Cardinality(\{nodes[n].version : n \in Nodes\}) \in 1 \,..\, 2
\end{array}
$$

Run the command to initialize an upgrade

$$
\begin{array}{l}
RunInitialize \triangleq \\
\quad \land upgradeState = Inactive \\
\quad \land \forall\, n \in Nodes : nodes[n].version = upgradeVersion \\
\quad \land upgradeState' = Initialized \\
\quad \land \text{UNCHANGED } \langle upgradeVersion,\, nodes \rangle
\end{array}
$$

Run the command to perform an upgrade

$$
\begin{array}{l}
RunUpgrade \triangleq \\
\quad \land upgradeState = Initialized \\
\quad \land CanUpgrade(upgradeVersion) \\
\quad \land Cardinality(\{n \in Nodes : nodes[n].version = NextVersion(upgradeVersion)\}) > 0 \\
\quad \land upgradeState' = Upgraded \\
\quad \land upgradeVersion' = NextVersion(upgradeVersion) \\
\quad \land \text{UNCHANGED } \langle nodes \rangle
\end{array}
$$

Run the command to rollback an upgrade

$$
\begin{array}{l}
RunRollback \triangleq \\
\quad \land upgradeState = Upgraded \\
\quad \land Cardinality(\{n \in Nodes : nodes[n].version = PreviousVersion(upgradeVersion)\}) > 0 \\
\quad \land upgradeState' = RolledBack \\
\quad \land upgradeVersion' = PreviousVersion(upgradeVersion) \\
\quad \land \text{UNCHANGED } \langle nodes \rangle
\end{array}
$$

Run the command to commit a version change

$$
\begin{array}{l}
RunCommit \triangleq \\
\quad \land upgradeState = Upgraded \\
\quad \land \forall\, n \in Nodes : nodes[n].version = upgradeVersion \\
\quad \land upgradeState' = Inactive \\
\quad \land \text{UNCHANGED } \langle upgradeVersion,\, nodes \rangle
\end{array}
$$

Run the command to reset *ISSU*

$$
\begin{array}{l}
RunReset \triangleq \\
\quad \land upgradeState = RolledBack \\
\quad \land \forall\, n \in Nodes : nodes[n].version = upgradeVersion
\end{array}
$$

$\land\ upgradeState' = Inactive$
$\land$ UNCHANGED $\langle upgradeVersion,\ nodes\rangle$

Start a node
$StartNode(n)\ \triangleq$
$\land\ nodes[n].state = Stopped$
$\land\ nodes' = [nodes$ EXCEPT $![n].state = Started]$
$\land$ UNCHANGED $\langle upgradeState,\ upgradeVersion\rangle$

Stop a node
$StopNode(n)\ \triangleq$
$\land\ nodes[n].state = Started$
$\land\ nodes' = [nodes$ EXCEPT $![n].state = Stopped]$
$\land$ UNCHANGED $\langle upgradeState,\ upgradeVersion\rangle$

Upgrade a node
$UpgradeNode(n)\ \triangleq$
$\land\ \lor\ \land\ upgradeState = Initialized$
$\land\ nodes[n].state = Stopped$
$\land\ nodes[n].version = upgradeVersion$
$\lor\ \land\ upgradeState = Upgraded$
$\land\ nodes[n].version = PreviousVersion(upgradeVersion)$
$\land\ CanUpgrade(nodes[n].version)$
$\land\ nodes' = [nodes$ EXCEPT $![n].version = NextVersion(nodes[n].version)]$
$\land$ UNCHANGED $\langle upgradeState,\ upgradeVersion\rangle$

Downgrade a node
$DowngradeNode(n)\ \triangleq$
$\land\ upgradeState = RolledBack$
$\land\ nodes[n].state = Stopped$
$\land\ nodes[n].version = NextVersion(upgradeVersion)$
$\land\ nodes' = [nodes$ EXCEPT $![n].version = PreviousVersion(nodes[n].version)]$
$\land$ UNCHANGED $\langle upgradeState,\ upgradeVersion\rangle$

Crash a node in a single atomic step
$CrashNode(n)\ \triangleq$
$\land\ \lor\ \land\ CanRollback$
$\land\ upgradeState' = RolledBack$
$\land\ upgradeVersion' = PreviousVersion(upgradeVersion)$
$\lor\ \land\ \neg CanRollback$
$\land$ UNCHANGED $\langle upgradeState,\ upgradeVersion\rangle$
$\land$ UNCHANGED $\langle nodes\rangle$

Initial state predicate

$$Init \triangleq$$
$$\land\ upgradeState = Inactive$$
$$\land\ upgradeVersion = Head(Versions)$$
$$\land\ nodes = [n \in Nodes \mapsto [state \mapsto Started,\ version \mapsto Head(Versions)]]$$

Next state predicate

$$Next \triangleq$$
$$\lor\ RunInitialize$$
$$\lor\ RunUpgrade$$
$$\lor\ RunCommit$$
$$\lor\ RunRollback$$
$$\lor\ RunReset$$
$$\lor\ \exists\, n \in Nodes :$$
$$\quad \lor\ StopNode(n)$$
$$\quad \lor\ StartNode(n)$$
$$\quad \lor\ UpgradeNode(n)$$
$$\quad \lor\ DowngradeNode(n)$$
$$\quad \lor\ CrashNode(n)$$

\ * Modification History
\ * Last modified *Fri Jan* 26 13:10:09 *PST* 2018 by *jordanhalterman*
\ * Created *Mon Jan* 22 23:16:53 *PST* 2018 by *jordanhalterman*